

DII.3200.NT40.PG-1

**Defense Information Infrastructure (DII)
Common Operating Environment (COE)**

Version 3.2.0.0

**Programming Guide
(Windows NT 4.0)**

June 13, 1997

Prepared for:

Defense Information Systems Agency

Prepared by:

**Inter-National Research Institute (INRI)
12200 Sunrise Valley Drive, Suite 300
Reston, Virginia 20191**

Table of Contents

Preface	1
1. Writing Programs Using the COE Tools	3
1.1 Overview	3
1.2 Referenced Documents	3
2. Application Development Overview	5
2.1 Writing Your Application with the DII COE APIs	5
2.2 Building Your Application with the DII COE APIs	5
2.3 Running Your Application	5
3. Segment Development	7
3.1 Segment Layouts	7
3.2 Running the COE Tools From the Command Line	8
3.2.1 COEInstaller Runtime Tool	9
3.2.2 COE Developer Tools	9
3.2.2.1 CalcSpace	10
3.2.2.2 CanInstall	11
3.2.2.3 MakeInstall	12
3.2.2.4 TestInstall	14
3.2.2.5 TestRemove	15
3.2.2.6 TimeStamp	16
3.2.2.7 VerifySeg	17
3.2.2.8 VerUpdate	18
3.3 Building Your Segment	19
3.3.1 Identifying and Creating Required Subdirectories	19
3.3.2 Creating and Modifying Required Segment Descriptor Files	19
3.3.3 Installing a Segment	21
3.4 Customizing Your Segment	22
3.4.1 Adding Menu Items	22
3.4.2 Adding Icons	29
3.4.3 Reserving a Socket	30
3.4.4 Displaying a Message	31

Appendix A - Sample Segment Layout	33
A.1 Sample Account Group Segment Layout	33
A.2 Sample Software Segment Layout	35
Appendix B - Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette	37
B.1 Running VerifySeg Against the Sample Segment	38
B.2 Running TestInstall Against the Sample Segment	38
B.3 Running TestRemove Against the Sample Segment	39
B.4 Running MakeInstall Against the Sample Segment	39

List of Tables

Table 1. Segment Descriptor Files	20
Table 2. SegInfo Segment Descriptor Sections	21

List of Figures

Figure 1. Segment Directory Structure	7
---	---

Preface

The following conventions have been used in this document:

[HELVETICA FONT]	Used to indicate keys to be pressed. For example, press [RETURN].
Courier Font	Used to indicate entries to be typed at the keyboard, operating system commands, titles of windows and dialog boxes, file and directory names, and screen text. For example, execute the following command: A:\setup.exe
<i>Italics</i>	Used for emphasis.

This page intentionally left blank.

1. Writing Programs Using the COE Tools

1.1 Overview

This document provides an introduction to the capabilities of the Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 3.2.0.0 tools for the Windows NT Operating System Version 4.0. These tools consist of a set of runtime tools and a set of developer's tools.

This document has been designed to help developers start using the DII COE tools. This document explains the basic use of the tools, regardless of whether they are run from a menu or from the command line. The document consists of the following sections and appendices:

Section/Appendix	Page
Application Development Overview Provides an overview of how to write, build, and run an application.	5
Segment Development Discusses the different types of segments and the process of segment creation.	7
Sample Segment Layout Describes how to install sample segments, which can be used to test segment installation and execution.	35
Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette Provides examples of how to convert a segment to the <i>DII COE Integration and Runtime Specification</i> segment format, verify segment syntax, temporarily install and remove a segment, and load a segment onto an installation floppy diskette.	39

1.2 Referenced Documents

The following documents are referenced in this guide:

- C DII COE I&RTS:Rev 3.0, *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification Version 3.0*, January 1997
- C DII.3200.NT40.RG-1, *Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 3.2.0.0 Application Programmer Interface (API) Reference Guide (Windows NT 4.0)*, June 13, 1997.

This page intentionally left blank.

2. Application Development Overview

Developers may require access to public Application Programmer Interfaces (APIs) to ensure an application complies with the *DII COE Integration and Runtime Specification*. To use these public APIs, developers must (1) include the public `include` files with the DII COE tools header and (2) compile and link the application with the Developer's Toolkit `include` directory and libraries. Public APIs are documented in the *DII COE API Reference Guide (Windows NT 4.0)*.

2.1 Writing Your Application with the DII COE APIs

To access the DII COE tools through the provided APIs, you must include the following header in your application:

```
#include <DIITools.h>
```

The standard location for the Developer's Toolkit header is:

```
DII_DEV\include
```

2.2 Building Your Application with the DII COE APIs

To build your application with the DII COE APIs, you must link your application with the `COECom.lib`, `COESeg.lib`, `COETools.lib`, and `COEUserPrompts.lib` libraries, which are on the DII COE Developer's Toolkit floppy diskettes.

The standard location for the Developer's Toolkit libraries is:

```
DII_DEV\libs
```

To compile the application, make sure the `include` file path in the compile environment includes `<local path>\DII_DEV\include`. Also make sure the library (`lib`) path includes `<local path>\DII_DEV\libs`.

2.3 Running Your Application

The DII COE provides the foundation and infrastructure in which one or more applications run. To operate under the COE, applications must be formatted properly as segments. The segment is the basic building block of the COE runtime environment. A segment is a collection of one or more Computer Software Configuration Items (CSCIs) that are managed most conveniently as a unit. Segments generally are defined to keep related CSCIs together so functionality easily may be included or excluded. All applications must be put in the DII COE runtime environment segment format to be installed onto a DII COE-compliant machine. Reference Section 4, *Segment Development*, for more information about segment development.

Once an application has been put in the proper segment format, the segment can be installed in a disciplined way through instructions contained in files provided with each segment. These files are called segment descriptor files and are contained in a special subdirectory, `SegDescrip`, which is called the segment descriptor subdirectory. Installation tools process the segment descriptor files to create a carefully controlled approach to adding or deleting segments to or from the system.

Once installed, your application can be invoked in the DII COE environment in two ways:

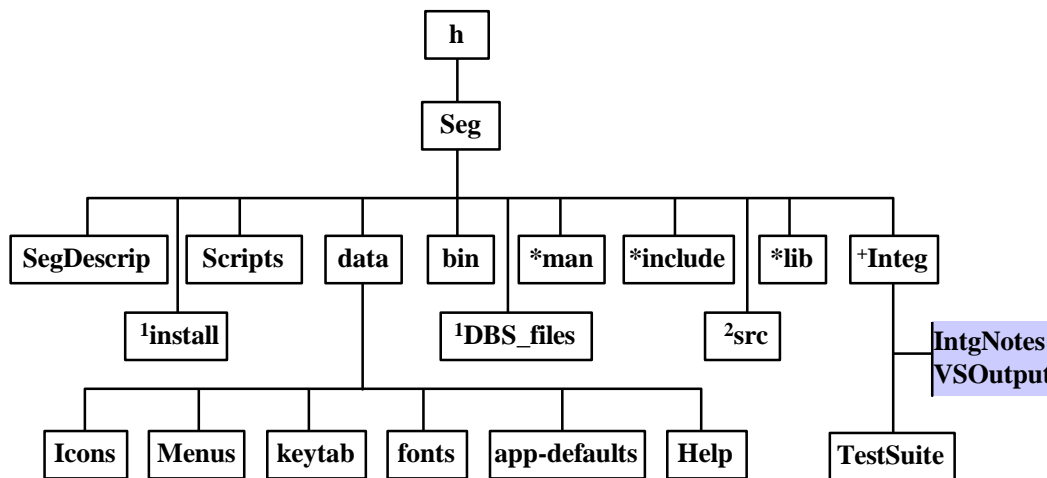
(1) running your application from a command shell window or (2) invoking your application from an icon. The easiest way to test your application is to invoke it in a command shell window. This gives you easy access to your application for debugging purposes and allows you to check any diagnostic information your application is generating. Section 3.4, *Customizing Your Segment*, describes how to set up your application to be invoked as a menu item or as an icon.

3. Segment Development

The following subsection discusses the different types of segments and the process of segment creation. Reference Section 5.0, *Runtime Environment*, of the *DII COE Integration and Runtime Specification* for a more detailed explanation of segments.

3.1 Segment Layouts

In the DII COE approach, each segment is assigned a unique, self-contained subdirectory. DII COE compliance mandates specific subdirectories and files underneath a segment directory. These subdirectories and files are shown in Figure 1. Six segment types exist: Account Group, COTS (Commercial Off-the-Shelf), Data, Database, Software, and Patch. The precise subdirectories and files required depend on the segment type. Some of the subdirectories shown in Figure 1 are required only for segment submission and are not delivered to an operational site.



- * Required for segments with published APIs
- + Required for segment submission
- ¹ For Database segments only
- ² Recommended location for source code during development,
Required location for source code delivered to DISA.

Figure 1. Segment Directory Structure

The following runtime subdirectories are normally required, depending on the segment type: (1) *SegDescrip*, which is the directory containing segment descriptor files; (2) *bin*, which is the directory containing executable programs for the segment; and (3) *data*, which is the subdirectory containing static data items, such as menu items, that are unique to the segment, but that will be the same for all users on all workstations.

The `SegDescrip` directory is required for every segment because it contains the installation instructions for the segment. A segment cannot modify files or resources outside its assigned directory. Files outside a segment's directory are called community files. COE tools coordinate modification of all community files at installation time. Reference Section 5.5, *Segment Descriptors*, of the *DII COE Integration and Runtime Specification* for a detailed explanation of `SegDescrip` files.

3.2 Running the COE Tools From the Command Line

The COE tools were constructed to aid developers in the creation and ultimate installation of DII COE segments. All tools can be run from the command line.

This section provides a brief overview of running the COE tools from the command line. When run from the command line, the tools are designed to run interactively and accept one or more command line parameters.

The tools are used to communicate with the outside world in two ways. First, the tools use the `RETURN` function to set the DOS `status` variable. The `status RETURN` value is set to 0 for normal tool completion or to 255 if an error occurs. A `status RETURN` value greater than 0 but less than 255 indicates a completion code that is tool specific. The `ERRORLEVEL` function may be used to examine a tool's exit status.

Second, the tools use `stdin` and `stdout` and thus support input and output redirection. Redirecting `stdin` allows the tools to receive input from a file or from another program, while redirecting `stdout` allows the tools to provide output to other programs.

NOTE: Redirecting `stdin` is not always convenient. The `-R` command line parameter allows a tool to read input from a response file instead of from `stdin`.

For example, the following statement can be used to write the results of `VerifySeg` to a file:

```
VerifySeg -p d:\testsegs Seg1 > results.txt
```

3.2.1 COEInstaller Runtime Tool

This section describes the COEInstaller tool, which is available at runtime. This executable is delivered to the operational site and is located underneath the h\COE\bin directory.

USAGE

COEInstaller [flags]

USABLE FLAGS

-h, -H	Display this help message.
-C <file>	Read command line arguments from the named <file>.
-d	Set the debug flag.
-v	Show verbose messages while the tool runs.
-V	Display the tool's version number.
-w	Suppress all warnings.

This tool displays a list of configuration definitions or segments that may be installed from floppy diskette or disk (e.g., a network segment server). The `status` environment variable is set to 0 if all requested segments were installed correctly or to -1 if any segment requested was not installed. By default, this tool does not write any output to `stdout`. This tool writes information to a status log that indicates installation progress, which segments have been installed, and other information that might be useful to the site administrator.

3.2.2 COE Developer Tools

This section lists the COE tools that are available during development but that are not delivered to operational sites. By default, these executables are located underneath the `DII_DEV` directory and are distributed as part of the Developer's Toolkit. These tools are not location sensitive and may be moved to any directory desired for development. For example, if the toolkit is tarred to \h, the path would be \h\DII_DEV.

All of the tools in this section are executed from a command line. MakeInstall is the only tool that has a GUI interface.

3.2.2.1 CalcSpace

USAGE

CalcSpace [flags] <segdir>

Where <segdir> is the home directory of the segment.

USABLE FLAGS

-h, -H	Display this help message.
-p <path>	Use <path> as the absolute path to the segment.
-v	Show verbose messages while the tool runs and print the space requirements for each top-level directory.
-V	Display the tool's version number.
-w	Suppress all warnings.

This tool computes the amount of space (in K bytes) that a segment is using. The segment should not be compressed in any way. The calculated value is written in the `Hardware` descriptor. Warnings will be issued for directories that are found but not expected (e.g., `include`, `src`) or if expected directories are missing (e.g., `bin` for a software segment). The total space calculated will be printed to `stdout`. The reserve space portion of the `Hardware` descriptor is not affected nor are any specified partitions. The `status` environment variable is set to 0 upon completion or to -1 if the `segdir` specified does not exist or does not appear to be a segment.

NOTE

<path> length and <segdir> length must be <= 256.

3.2.2.2 CanInstall

USAGE

CanInstall [flags] <segdir>

Where <segdir> is the home directory of the segment.

USABLE FLAGS

-h, -H	Display this help message.
-p <path>	Use <path> as the absolute path to the segment.
-v	Show verbose messages while the tool runs.
-V	Display the tool's version number.
-w	Suppress all warnings.

This tool tests a segment to see if it can be installed. It performs the same tests that the COEInstaller tool performs at installation time. To be installable, all required segments already must be on the disk. In addition, the disk must not contain any conflicting segments. The `status` environment variable is set to 0 if the segment can be installed or to -1 if the segment cannot be installed. An error message is printed to indicate why a segment cannot be installed.

3.2.2.3 MakeInstall

USAGE

MakeInstall [flags] <segname>

Where <segname> is a list of one or more segments to process.

USABLE FLAGS

-h, -H	Display this help message.
-C <file>	Read command line arguments from the named <file>.
-Cd <exe> <path>	Use <exe> as the name of the executable that will check the segment descriptor out of the repository and place it in the temporary location indicated by absolute path name <path>. The location of the executable is determined by the most recent occurrence of the -p flag.
-Co <exe> <path>	Use <exe> as the name of the executable that will check the segment out of the repository and place it in the temporary location indicated by the absolute path name <path>. The location of the executable is determined by the most recent occurrence of the -p flag.
-d on off	Specify whether to delete (-d on) or not to delete (-d off) segments checked out of the repository when MakeInstall is completed.
-di <definitions>	Use <definitions> as a list of one or more distribution definition files as created by ConfigDef. The location of the distribution is determined by the most recent occurrence of the -p flag. These distributions are not in a repository. All of the segments that comprise the distribution must be located by the -p flag or through the directory list in the \$SEG_DIRS path environment variable or else an error will be generated.
-dio <definitions>	Use <definitions> as a list of one or more distribution definition files as created by ConfigDef. The definitions are to be checked out of the repository by the program provided by the -Co and -Cd flags. The segments that comprise the distribution also must be included (via -s, -so, -S, or -So) or an error will be generated.
-Di <files>	Use <files> as a list of one or more files that contain a list of distribution definitions to write to the output medium. The location of the file is determined by the most recent occurrence of the -p flag, while the distribution definitions are located by the \$SEG_DIRS environment variable or by -p flags in the list file. These distributions are not in a repository. All of the segments that comprise the distribution must be located by the -p flag or through the directory list in the \$SEG_DIRS path environment variable or else an error will be generated.
-Dio <files>	Use <files> as a list of one or more files that contain a list of distribution definitions to write to the output medium. The list of distribution definitions are to be checked out of the repository specified by the -Co and -Cd flags. All of the segments that comprise the variant also must be included (via -s, -so, -S, or -So) or an error will be generated.
-f	Echo segment names as they are processed (default is OFF).
-o <file>	Use the disk as the output medium instead of floppy diskette.

<code>-ot <files></code>	Use <code><files></code> as a list of one or more segment files created by MakeInstall with the <code>-o</code> flag. This flag writes files to the output medium in a format suitable for installation by the COEInstaller tool.
<code>-p <path></code>	Use <code><path></code> to establish a path to segments, variants, and so forth.
<code>-R <file></code>	Use <code><file></code> to respond to questions from the tool.
<code>-s <segs></code>	Use <code><segs></code> as a list of one or more segments to write to the output medium. The location of the segments is determined by most recent occurrence of the <code>-p</code> flag. These segments are not in a repository.
<code>-so <segs></code>	Use <code><segs></code> as a list of one or more segments to write to the output medium. The segments are to be checked out of the repository by the program provided by the <code>-Co</code> and <code>-Cd</code> flags.
<code>-S <files></code>	Use <code><files></code> as a list of one or more files that contain a list of segments to write to the output medium. The location of the files is determined by the most recent occurrence of the <code>-p</code> flag. Segments are located under <code>\h</code> or as determined by the <code>-p</code> flag within the file. These segments are not in a repository.
<code>-So <files></code>	Use <code><files></code> as a list of one or more files that contain a list of segments to be checked out of the repository specified by the <code>-Co</code> and <code>-Cd</code> flags.
<code>-t <dev></code>	Use <code><dev></code> as the output device (e.g., <code>/dev/rmt/3m</code>).
<code>-T</code>	Read and display the floppy diskette's table of contents.
<code>-v</code>	Show verbose messages while the tool runs.
<code>-V</code>	Display the tool's version number.
<code>-w</code>	Suppress all warnings.
<code>-x</code>	Validate but do not actually create a floppy diskette.

This tool writes one or more segments to a floppy diskette and packages the segments for distribution over the network. MakeInstall checks if VerifySeg has been run successfully on each of the segments and aborts with an error if it has not. The `status` environment variable is set to `-1` if any errors occur and to `0` if the process is successful. Error messages are written to `stdout` as appropriate.

NOTE

The flags `-Cd`, `-Co`, `-dio`, `-Dio`, `-ot`, `-R`, `-so`, `-So`, and `-T` are not currently supported.

NOTE

If no path is specified, `/h` will be used.

3.2.2.4 TestInstall

USAGE

TestInstall [flags] <segment_list>

USABLE FLAGS

-h, -H	Display this help message.
-C <file>	Read command line arguments from the named <file>.
-e	Echo descriptor contents as they are processed. Enabling this flag thereby enables the -f flag.
-f	Echo descriptor names as they are processed.
-p <path>	Use <path> as the absolute path to the source directory.
-v	Show verbose messages while the tool runs.
-V	Display the tool's version number.
-w	Suppress all warnings.

This tool is used to temporarily install a segment that already resides on disk. It must be run when no other COE processes are running. The reason for this restriction is that the tool may modify COE files already in use with unpredictable results. VerifySeg must have been run before TestInstall to make sure that the segment is valid.

This tool performs the same operations as the COEInstaller tool except that it does not need to read the segment from floppy diskette (e.g., it is already on disk), and the segment may be in any arbitrary location. This tool will establish a registry entry for the segment. The `status` environment variable is set to 0 if the installation is successful or to -1 if the installation is not successful. Error and status messages are written to `stdout` as required. The TestInstall tool must be run as Administrator because it modifies files the user may not own.

3.2.2.5 TestRemove

USAGE

TestRemove [flags] <segname>

Where <segname> is the name of the segment to be removed.

USABLE FLAGS

-h, -H	Display this help message.
-C <file>	Read command line arguments from the named <file>.
-e	Echo descriptor contents as they are processed--enabling this flag thereby enables the -f flag.
-f	Echo descriptor names as they are processed.
-p <path>	Use <path> as the source directory.
-v	Show verbose messages while the tool runs.
-V	Display the tool's version number.
-w	Suppress all warnings.

This tool is used to remove a segment that was installed by TestInstall. It must be run when no other COE processes are running. The reason for this restriction is that the tool may modify COE files already in use with unpredictable results. This tool removes the registry entry if one exists, but it does *not* delete the segment from disk. The `status` environment variable is set to 0 if the removal is successful or to -1 if the removal is not successful. Error and status messages are written to `stdout` as required. The TestRemove tool must be run as Administrator because it modifies files the user may not own.

NOTE

This tool is unconditional and should be used with great caution. It will remove a specified segment even if other installed segments still depend on it.

3.2.2.6 TimeStamp

USAGE

TimeStamp [flags] <segname>

USABLE FLAGS

-h, -H	Display this help message.
-p <path>	Use <path> to establish a path for subsequent file names.
-V	Display the tool's version number.

This tool puts the current time and date into the `VERSION` segment descriptor. It is intended to assist the configuration process by allowing the time stamp to be updated just before running `VerifySeg` and `mkSubmitTar` for the deliverable product. The `status` environment variable is set to `-1` if an error occurs (e.g., the `VERSION` segment descriptor is not found) or to `0` if the time stamp is successful. `TimeStamp` should be run as the `root` user, although it is not mandatory. This tool requires the user to have write permission to the segment against which the tool was executed.

3.2.2.7 VerifySeg

USAGE

VerifySeg [flags] <segdir> ...

Where <segdir> is the home directory of the segment to be verified.

USABLE FLAGS

-h, -H	Display this help message.
-C <file>	Read command line arguments from the named <file>.
-e	Echo descriptor lines as they are processed.
-f	Echo descriptor names as they are processed.
-o	Identify obsolete usage in the segment.
-p <path>	Use <path> to establish a path for subsequent file names.
-R <file>	Use responses listed in <file> to answer questions.
-s <name>	Only validate the descriptor <name>.
-t	Print a table of required/optional descriptors.
-v	Show verbose messages while the tool runs.
-V	Display the tool's version number.
-w	Suppress all warnings.
-x <name>	Display syntax for the descriptor <name>.

This tool is used to validate that a segment conforms to the rules for defining a segment. It uses information in the `SegDescrip` subdirectory and must be run whenever the segment is modified. This tool must be run for each segment. If the segment is an aggregate segment, it must be run on each segment in the aggregate. The status environment variable is set to -1 if any errors occur or to 0 if the validation is successful. Error and status messages are written to `stdout` as required. VerifySeg should be run as `Administrator`, although it is not mandatory. This tool requires the user to have write permission to the segment against which the tool was executed.

NOTE

The -p flag and <SegDir> can be specified more than once on the command line.

3.2.2.8 VerUpdate

USAGE

VerUpdate [flags] <segname>

USABLE FLAGS

-h, -H	Display this help message.
-v	Display the tool's version number.
-p <path>	Use <path> to establish a path for subsequent file names.
-i <version>	Use <version> as the version number to insert unconditionally.
-d <digit>	Use <digit> to increment the digit specified by one in the version number. For example, the version number <1.2.3.4>. <digit> is 1 for the major release digit <digit> is 2 for the minor release digit <digit> is 3 for the maintenance release digit <digit> is 4 for the developer digit
-ip <version>	Use <version> as the patch version number to insert unconditionally.
-ap	Add path version number 'P1' to the VERSION file if it does not exist already.
-up	Increment the patch version number by one if it exists.

This tool updates the segment version number, current time, and current date in the VERSION descriptor. It is intended to assist the configuration process by allowing the version, current time, and current date to be updated just before running VerifySeg and mkSubmitTar for the deliverable product. The status environment variable is set to 0 if the version update is successful or to -1 if an error occurs (e.g., the segment was not found). VerUpdate should be run as the root user, although it is not mandatory. This tool requires the user to have write permission to the segment against which the tool was executed.

NOTE

Command line parameters -ip, -ap, and -up may only be used one at a time.

3.3 Building Your Segment

A segment must be built in a disciplined way using instructions contained in files provided with each segment. These files are contained in a special directory, `SegDescrip`, which is the segment descriptor subdirectory.

This section describes a process to turn an application into a segment so it can be a part of the DII COE. As described earlier, a segment is a collection of one or more CSCIs most conveniently managed as a unit.

3.3.1 Identifying and Creating Required Subdirectories

There are six segment types: Account Group, COTS, Data, Database, Software, and Patch. The following subdirectories normally are required:

Subdirectory	Description
<code>SegDescrip</code>	Subdirectory containing segment descriptor files. This directory is always required for every segment and contains the installation instructions for the segment. A segment is not allowed to directly modify any files for resources it does not <i>own</i> ; in other words, a segment cannot modify files or resources outside an assigned directory. The DII COE tools coordinate the modification of all community files at installation time.
<code>bin</code>	Executable programs for the segment. These files can be the result of a compiled program or as a result of shell scripts, depending on the type of the segment.
<code>data</code>	Subdirectory for static data items, such as menu items, that are unique to the segment, but that will be the same for all users on all workstations.

Reference Sections 5.0-5.5 of the *DII COE Integration and Runtime Specification* for a detailed explanation of segment directory layout and a description of each `SegDescrip` file.

3.3.2 Creating and Modifying Required Segment Descriptor Files

Segment descriptor files are the key to providing seamless and coordinated systems integration across all segments. Reference Table 1 to determine which descriptor files are required for each segment type. For example, the `AcctGrp` segment requires `ReleaseNotes`, `SegInfo`, `SegName`, and `VERSION` descriptor files in the `SegDescrip` directory, while the `Patch` segment requires the `PostInstall` descriptor file in addition to the previously listed files. Some segment descriptor information is provided in the files listed in Table 1.

NOTE: In Table 1, `Aggregate` and `COE Comp` are segment attributes that can be associated with any type of segment.

File	Acct Grp	COTS	Data	DB	S/W	Patch
------	-------------	------	------	----	-----	-------

DEINSTALL	O	O	O	O	O	O
FileAttribs	O	O	O	O	O	O
Installed	I	I	I	I	I	I
PostInstall	O	O	O	O	O	R
PreInstall	O	O	O	O	O	O
PreMakeInst	O	O	O	O	O	O
ReleaseNotes	R	R	R	R	R	R
SegChecksum	I	I	I	I	I	I
SegInfo	R	R	R	R	R	R
SegName	R	R	R	R	R	R
Validated	I	I	I	I	I	I
VERSION	R	R	R	R	R	R
R - Required O - Optional N - Not Applicable I - Created by Integrator or Installation Software						

Table 1. Segment Descriptor Files

Other segment descriptor information is arranged within subsections of the `SegInfo` file. As with the descriptor files themselves, some subsections of the `SegInfo` file are required and others are optional depending on the type of segment. Table 2 defines required and optional sections for each segment type.

Section	Acct Grp	COTS	Data	DB	S/W	Patch
AcctGroup	R	N	N	N	N	N
AppPaths	O	N	N	N	O	N
COEServices	O	O	O	O	O	O
Community	O	O	O	O	O	O
Comm.deinstall	O	O	O	O	O	O
Compat	O	O	O	O	O	N
Conflicts	O	O	O	O	O	O
Data	N	N	R	N	N	N
Database	N	N	N	R	O	O
Direct	O	O	O	O	O	O
FilesList	O	R	O	O	O	O
Hardware	R	R	R	R	R	R
Help	O	O	O	O	O	O
Icons	R	O	N	N	O	O
Menus	R	O	N	N	O	O
Network	N	N	N	N	N	N
Permissions	O	N	N	N	O	O
Processes	O	O	N	N	O	O
Registry	O	O	O	O	O	O
Requires	O	O	O	O	O	O
Security	R	R	R	R	R	R
SharedFile	O	O	N	N	O	O
R - Required O - Optional N - Not Applicable						

Table 2. SegInfo Segment Descriptor Sections

3.3.3 Installing a Segment

Follow the procedures below to install a segment after it has been created.

Run VerifySeg

The VerifySeg tool must be run during the development phase to ensure segments use segment descriptor files properly. Run the VerifySeg tool whenever a segment is created or modified. When VerifySeg is run to verify a segment, a `Validated` file is created. Reference Section 3.2.2.9, *VerifySeg*, for further information about using VerifySeg.

Run TestInstall

Executing the TestInstall tool is not a mandatory step in the installation process, but it is recommended. TestInstall simulates an installation of a segment on the developer's workstation before actual installation. Reference Section 3.2.2.6, *TestInstall*, for further information about using TestInstall.

Run TestRemove

Executing the TestRemove tool is not a mandatory step in the installation process, but it is recommended. TestRemove simulates a deinstallation of a segment on the developer's workstation after TestInstall has been run. Reference Section 3.2.2.7, *TestRemove*, for further information about using TestRemove.

Run MakeInstall

The MakeInstall tool is used to write one or more segments to an installation media and to package the segment(s) for distribution. MakeInstall checks if VerifySeg has been run successfully on each of the segments and aborts with an error if it has not. Reference Section 3.2.2.4, *MakeInstall*, for further information about using MakeInstall.

Run COEInstaller

The COEInstaller tool installs a segment from floppy diskette. Reference Section 3.2.1, *COEInstaller Runtime Tool*, for further information about using COEInstaller.

3.4 Customizing Your Segment

Most properly designed segments will not require any extensions to the COE, although the segments may need to add menu items and icons. Some segments may need to add special extensions. The following subsections describe how to add menu items, icons, and special extensions.

3.4.1 Adding Menu Items

Menu files are maintained by the DII COE, but no DII COE applications read menu files in this release. Nevertheless, user programs may use their own menu files through this feature.

Menu Entry Format

The Menu Descriptor in the `SegInfo` file is used to specify the name of the segment's menu file and the name of the affected segment's menu file.

The menu bar, pull-down menus, and cascade menus, as well as the menu items they contain, are built according to the menu description entries. The format of the entries is in ASCII with colon-separated fields. Colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A `#` symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the entry and are not processed by the parser.

Valid keywords are `PDMENU`, `PDMENUEND`, `ITEM`, `PRMENU`, `CASCADE`, `CASCADEEND`, `APPEND`, `APPENDEND`, and `SEPARATOR`. Since the Menu Description Entry is based on your menu design, you might not use all of these keywords. For example, if your menu does not have separator lines, your Menu Description Entry will not contain a `SEPARATOR` keyword.

Each keyword is described in the following paragraphs.

A **`PDMENU`** line contains the following elements:

```
PDMENU: name : enable flag : id # :
```

<code>PDMENU</code>	Keyword that indicates the start of a pull-down menu.
<code>name</code>	Text used to name the menu. The menu name is displayed on the menu bar.
<code>enable flag</code>	Integer value that indicates whether a menu is enabled or disabled. The enable flag is <code>1</code> if a menu is enabled or <code>0</code> if it is disabled. A disabled menu means that no options under that pull-down menu can be selected.

id# Optional integer value that provides a unique ID number for the menu. The `PDMENU id#` value must be unique within the menu description file. An absolute value may be provided. However, the `id#` field should be left empty so that relative numbering is used by default.

With relative numbering, an `id#` of `R1` (or leaving the field blank) sets the menu's ID number to 1 plus the `id#` of the last menu processed. An `id#` of `R2` sets the menu's ID number to 2 plus the `id#` of the last menu processed.

The following is an example of a `PDMENU` line:

```
PDMENU: Map Options : 1 : R1 :
```

A **`PDMENUEND`** line contains the following element:

```
PDMENUEND:
```

`PDMENUEND` Optional keyword that indicates the end of a group of pull-down menu items. If `PDMENUEND` is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than `ITEM` or `PRMENU`) is encountered.

The following is an example of a `PDMENUEND` line:

```
PDMENUEND:
```

An **`ITEM`** line contains the following elements:

```
ITEM: name : command : execution type : enable flag : # instances : id# :  
check value : security char : autolog flag : print flag : disk flag :
```

`ITEM` Keyword that indicates a menu item description line.

name Text used to name the menu item. The item name is displayed in the pull-down menu.

command Program with space-separated arguments that is launched if the menu item type is a program. Otherwise, the menu item is called as an application callback. Because callback functions must be linked into the same executable as the menu bar, applications cannot use callbacks when adding items to the system menu bar.

<i>execution type</i>	<p>Integer value that indicates how to execute a command, as follows:</p> <ul style="list-style-type: none">1 = executable program2 = void callback function with no parameters (not yet implemented)3 = Motif callback function (not yet implemented).
<i>enable flag</i>	<p>Integer value that indicates if a menu item is enabled or disabled. The enable flag is 1 if a menu item is enabled or 0 if it is disabled. A disabled menu item means that the option cannot be selected.</p>
<i># instances</i>	<p>Integer value used to set the maximum number of times the item can be executed simultaneously.</p>
<i>id#</i>	<p>Optional integer value that provides a unique ID number for the menu item. Each <code>ITEM id#</code> entry must be unique within a <code>PDMENU</code> listing. (<code>ITEM</code> entries in a <code>PRMENU</code> must be unique within that <code>PRMENU</code>.) Reference the <code>id#</code> description under the <code>PDMENU</code> keyword listing.</p>
<i>check value</i>	<p>Optional integer value that sets the star and checks annotations of a menu item. Possible values are:</p> <ul style="list-style-type: none">0 = no annotation (default)1 = visible check mark2 = check mark, but not visible3 = visible star4 = star member, but not visible.

This element is not yet fully implemented.

<i>security char</i>	<p>Optional character value used to determine the lowest security level under which a menu item can be classified. Valid settings are:</p> <ul style="list-style-type: none">N = No ClassificationU = Unclassified (default)C = ConfidentialS = SecretT = Top Secret.
<i>autolog flag</i>	<p>Optional character value, <code>T</code> or <code>F</code>, used to indicate if the command should be logged automatically. This element is not yet fully implemented.</p>
<i>print flag</i>	<p>Optional character value, <code>T</code> or <code>F</code>, used to indicate if the command should have a print capability. This element is not yet fully implemented.</p>

`disk flag` Optional character value, T or F, used to indicate if the command should have a disk access capability. This element is not yet fully implemented.

The following is an example of an `ITEM` line:

```
ITEM: Netscape : Netscape.. : 1 : 1 : 1 : R1 : 0 : T : F : F : F :
```

A **PRMENU** line contains the following elements:

```
PRMENU: name : enable flag : id# :
```

<code>PRMENU</code>	Keyword that indicates a cascading menu button. It is used to mark where a cascade menu is to be connected to an upper-level menu.
<code>name</code>	Text used to name the cascade menu with which to connect. The <code>PRMENU</code> name is displayed in the pull-down menu.
<code>enable flag</code>	Integer value that indicates if a cascade menu is enabled or disabled. The enable flag is 1 if a cascade menu is enabled or 0 if it is disabled. A disabled cascade menu means that menu options on the cascade menu cannot be selected.
<code>id#</code>	Optional integer value that provides a unique ID number for the cascading menu. Each <code>PRMENU id#</code> must be unique within a <code>PDMENU</code> listing. Reference the <code>id#</code> entry under the <code>PDMENU</code> keyword listing.

The following is an example of a `PRMENU` line:

```
PRMENU: Software : 1 : R1 :
```

A **CASCADE** line contains the following element:

```
CASCADE: name :
```

<code>CASCADE</code>	Keyword that indicates the start of a cascade menu. The cascade menu connects to the <code>PRMENU</code> entry of the same name.
<code>name</code>	Text used to name a cascade menu. The name is used to attach a cascade menu to a cascading menu button. This name must be the same as the name field in the <code>PRMENU</code> entry.

The following is an example of a `CASCADE` line:

```
CASCADE: Software :
```

A **CASCADEEND** line contains the following element:

```
CASCADEEND:
```

<code>CASCADEEND</code>	Optional keyword that indicates the end of a group of cascade menu items. If <code>CASCADEEND</code> is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than <code>ITEM</code> or <code>PRMENU</code>) is encountered.
-------------------------	--

The following is an example of a `CASCADEEND` line:

```
CASCADEEND:
```

An **APPEND** line contains the following elements:

```
APPEND: name :
```

<code>APPEND</code>	Keyword that indicates the start of a group of items to append to an existing menu. The menu will be created if it does not already exist. The group is appended to the <code>PDMENU</code> or <code>CASCADE</code> entry of the same name.
---------------------	---

<code>name</code>	Text used to select the menu to which a group of items is appended.
-------------------	---

The following is an example of an `APPEND` line:

```
APPEND: Options :
```

An **APPENDEND** line contains the following element:

```
APPENDEND:
```

<code>APPENDEND</code>	Optional keyword that indicates the end of a group of menu items to be appended to an existing menu. If <code>APPENDEND</code> is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than <code>ITEM</code> or <code>PRMENU</code>) is encountered.
------------------------	--

The following is an example of an `APPENDEND` line:

```
APPENDEND:
```

A **SEPARATOR** line contains the following element:

SEPARATOR:

SEPARATOR Optional keyword that indicates that a Motif separator widget is to be placed in a menu at the point where the keyword occurs.

The following is an example of a **SEPARATOR** line:

SEPARATOR:

Example of Adding a Menu Item

To add menu items, include the **Menus Descriptor** in the **SegInfo Segment Descriptor** file. Specify the **Menu** file you use wish to load and the **Menu** file you wish to update. The **Menu** file you wish to load should be located in the **TstSeg\data\Menu** directory, assuming the segment name is **TstSeg**. The following example will add the **Test Program** menu item to the **Software** menu under the **SysAdm** account group.

The program **TSTCOEAskUser_example** will be executed when invoked through the menu item:

SegInfo (menu descriptor only)

```
[Menus]
TstSegMenu:SA_Default.main
TstSegMenu
#-----
# Software Menu Items
#-----
APPEND           :Software
ITEM             :Test Program   :TSTCOEAskUser_example:1:1:1:R1
APPENDEND :
```

The **\$SEGMENT** keyword must be used in the **SegName Segment Descriptor** file to specify the name of the affected segment. In this case it is **System Administration**.

SegName

```
#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

3.4.2 Adding Icons

Icon Entry Format

The Icon Description Entry contains information on all icon-based processes. The entry, or set of entries, to be used is passed to the Program Manager.

Icons are built using the icon section in the `SegInfo` file. The entry is a specially formatted icon description that has colon-separated fields. The colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A # symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the file and are not processed by the parser.

The format of the icon entry is as follows:

```
ICON file : affected icon file
```

The affected icon file contains information about both the icon and the executable. The format of the file is as follows:

```
Icon Name : Icon File : Executable File : Comments
```

Where `Icon Name` is the title placed next to the icon in the `Programs` menu, `Icon File` is the icon image file, `Executable File` is the executable to be launched by the Program Manager, and `Comments` is the comment line. The `Icon File` field is optional. If an `Icon File` is specified, the file must be located in the segment's `data\Icons` directory. The `Executable File` must be located in the segment's `bin` directory.

An example of an affected icon file is as follows:

```
Edit Profiles : EditProf.ico: EditProf.exe
```

Icons are added to a `Programs` menu group named for the account group to which they belong.

Example of Adding an Icon

To add an icon, include the `Icons` Descriptor in the `SegInfo` Segment Descriptor file. Specify the icon file you wish to load. The icon file you wish to load should be located under the `TstSeg\data\Icons` directory, assuming the segment directory is `TstSeg`. This example will add the `Test Program` icon to the `SysAdm` account group. When invoked through the icon, the program `TSTCOEAskUser_example` will be executed.

SegInfo (icon descriptor only)

```
[Icons]
TstSegIcons:SA_Default
```

TstSegIcons

```
#-----
# Software Icons
#-----
Test Program :TestProgramIcon:TSTCOEAskUser_example
```

The \$SEGMENT keyword must be used in the SegName Segment Descriptor file to specify the name of the affected segment. In this case it is System Administration

SegName

```
#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:SystemAdministration:SA:/h/AcctGrps/SysAdm
```

3.4.3 Reserving a Socket

To add a service, include the COEServices Descriptor in the SegInfo Segment Descriptor file. Also include the \$SERVICES keyword in the SegInfo Segment Descriptor file to specify the service to be added. If the port number requested is already in use under another name, an error will be generated.

NOTE: Port numbers in the range 2000-2999 are reserved for DII COE segments.

SegInfo (COEServices descriptor only)

```
[COEServices]
#
# This is my service to add
#
$SERVICES
irc_ser:3001:upd
```

3.4.4 Displaying a Message

This subsection shows an example of how to display a message during the PostInstall process. Five runtime tools can be used to communicate with the user: COEAskUser, COEInstError, COEMsg, COEPrompt, and COEPromptPasswd. These tools may be used to display information to the user or to ask the user a question and, based on the result, perform different actions.

In this example, the user is asked questions using the COEAskUser runtime tool, which is described in Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification*.

```
rem =====
rem PostInstall script for Segment Tst
rem =====

echo "Performing PostInstall"

COEAskUser -B "RED LAN" "BLUE LAN" "Which LAN will you be connecting to?"

if ERRORLEVEL 1 goto RED

if ERRORLEVEL 0 goto BLUE

goto END:

:RED
echo "Connecting to RED LAN"
rem
rem Perform some action based user input
rem
goto END:

:BLUE
echo "Connecting to BLUE LAN"
rem
rem Perform some action based on user input
rem

:END
echo "Done with PostInstall"
```

This page intentionally left blank.

Appendix A - Sample Segment Layout

This appendix includes a segment layout for a sample Account Group segment (GCCS) and a sample Software segment (Seg1). These basic templates of typical DII COE segments can be used to test segment installation and execution.

NOTE: If you do not have the GCCS sample Account Group segment installed on your machine, you will receive several warnings indicating that it must be installed before the Seg1 sample Software segment can be loaded.

The Seg1 sample Software segment will add the Seg1 Hello World icon to the GCCS Account Group in the Programs menu. The program Seg1_HelloWorld.exe will be executed when invoked through the icon.

Refer to Appendix B, *Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette*, for instructions on how to validate the Seg1 sample Software segment and load the segment onto a floppy diskette.

A.1 Sample Account Group Segment Layout

The layout of the GCCS sample Account Group segment is:

```
gccs
    SegDescrip
        DEINSTALL.BAT
        PostInstall.bat
        ReleaseNotes
        SegInfo
        SegName
        VERSION
```

The SegDescrip files contain the following:

```
DEINSTALL.BAT
REM =====
REM
REM DEINSTALL
REM
REM Routine to perform necessary actions when segment
REM is deinstalled.
REM
REM =====
```

PostInstall.bat

```
REM =====
REM
REM PostInstall
REM
REM Routine to perform necessary actions after segment
REM has been loaded.
REM
REM =====
```

ReleaseNotes

This is a sample Account Group.
Other Segments will need to extend the environment
as they add their specific functionality
to the account group.

SegInfo

```
#=====
#
#   Account Group SegInfo file.
#
#=====
```

```
[AcctGroup]
GCCS Operator:350::1:GCCS:GCCS Default
$CLASSIF:UNCLASS
```

```
[Hardware]
$CPU:PC
$OPSYS:NT
$DISK:500
$MEMORY:100
```

```
[Security]
UNCLASS
```

SegName

```
#=====
#
#   Account Group SegName file.
#
#=====
$TYPE:ACCOUNT GROUP
$NAME:GCCS COE
$PREFIX:GCCS
```

VERSION

3.2.0.0:2/5/97

A.2 Sample Software Segment Layout

The layout of the Seg1 sample Software segment is:

```
Seg1
  bin
    Seg1_HelloWorld.exe
  data
    Icons
      TestIcons
  SegDescrip
    DEINSTALL.BAT
    PostInstall.BAT
    ReleaseNotes
    SegInfo
    SegName
    VERSION
```

The SegDescrip files contain the following:

```
DEINSTALL
REM =====
REM
REM DEINSTALL
REM
REM Routine to perform necessary actions when Seg1
REM is deinstalled.
REM
REM =====
```

```
PostInstall.BAT
REM =====
REM
REM PostInstall
REM
REM Routine to perform necessary actions after Seg1 Test
REM Segment has been loaded.
REM
REM =====
```

```
ReleaseNotes
This is the Seg1 Test Segment.
```

SegInfo

```
#=====
#
#   DII Database Admin Segment SegInfo
#   Descriptor file.
#
#=====
[Hardware]
$CPU:PC
$OPSYS:NT
$DISK:500
$MEMORY:100

[Icons]
TestIcons

[Security]
UNCLASS
```

SegName

```
#=====
#
#   DII Seg1 Test Segment
#   Descriptor file.
#
#=====
$TYPE:SOFTWARE
$NAME:Test Segment #1
$PREFIX:Seg1
$SEGMENT:GCCS COE:GCCS:/h/AcctGrps/GCCS
```

VERSION

3.2.0.0:10/15/96

The data\Icons file contains the following:

TestSegIcons

```
TstSegIcons
#-----
# Software Icons
#-----
Test Program :TestProgramIcon.ico:TSTCOEAskUser_example
```

Appendix B - Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette

This appendix provides examples of how to verify segment syntax, install a segment temporarily, and load a segment onto an installation floppy diskette. The segment verification and loading process involves the following steps:

- STEP 1: Run the VerifySeg tool.** Run VerifySeg to validate that the segment conforms to the rules for defining a segment (i.e., to verify the segment syntax). Type the following command to run VerifySeg:

```
VerifySeg -p [segment path] [segment directory]
```

- STEP 2: Run the TestInstall tool.** Run TestInstall against the sample segment to install the segment temporarily. This step is optional; if you choose not to run TestInstall, proceed to STEP 4. Type the following command to run TestInstall:

```
TestInstall -p [segment path] [segment directory]
```

- STEP 3: Run the TestRemove tool.** Run TestRemove against the sample segment to remove the segment temporarily installed in STEP 2. Type the following command to run TestRemove:

```
TestRemove -p [segment path] [segment directory]
```

- STEP 4: Run the MakeInstall tool.** Run MakeInstall to load the segment onto a floppy diskette. Type the following command to run MakeInstall:

```
MakeInstall -p [segment path] [segment directory]
```

After the segment is loaded onto a floppy diskette, it is ready to be installed using the DII Installer icon from the System Administration group.

Subsections B.1-B.4 show how to perform these steps against the Seg1 sample Software segment, which is described in Appendix A, *Sample Segment Layout*.

NOTE: In the following subsections, the VerifySeg, TestInstall, TestRemove, and MakeInstall tools are being run against the Seg1 sample Software segment. The output of each command will vary depending on the segment being converted. Note the following severity indicators:

(F) indicates a FATAL ERROR
(W) indicates a WARNING
(E) indicates an ERROR

(D) indicates a DEBUG statement
(V) indicates a VERBOSE statement
(O) indicates an ECHO statement.

NOTE: In the following subsections, boldface text indicates information that the user must input.

B.1 Running VerifySeg Against the Sample Segment

VerifySeg -p \SampleSegs Seg1

Results of verification (/SampleSegs/Seg1) :

Totals

Errors: 0

Warnings: 0

B.2 Running TestInstall Against the Sample Segment

TestInstall -p \SampleSegs Seg1

TestInstall - Version 1.0.0.7

The following options have been selected:

Print warning messages.

Segments to be TestInstalled:

Segment: seg1 Path: P:\SampleSegs

*****WARNING*****

TestInstall may modify COE files already in use.

This may cause unpredictable results if COE processes are already running.

Make sure no other COE processes are running before using TestInstall.

Do you want to continue with the TestInstall? (y/n): **y**

Processing seg1

Successfully ran preprocessor on segment seg1

No PreInstall script for segment seg1

Do you want to run PostInstall for Segment seg1? (y/n): **y**

REM =====

REM

REM PostInstall

REM

REM Routine to perform necessary actions after Seg1 Test

REM Segment has been loaded.

REM

REM =====

Successful Installation of seg1

B.3 Running TestRemove Against the Sample Segment

```
*****
TestRemove -p \SampleSegs Seg1
*****WARNING*****
TestRemove may modify COE files already in use.
This may cause unpredictable results if COE processes are already running.
Make sure there are no other COE processes running before using TestRemove.
Do you want to continue with the TestRemove? (y/n):y
SETTING P:\SampleSegs\Seg1 FOR INSTALL_DIR

REM =====
REM
REM  DEINSTALL
REM
REM  Routine to perform necessary actions when Seg1
REM  is deinstalled.
REM
REM =====
*****
Successful Removal of Seg1
```

B.4 Running MakeInstall Against the Sample Segment

The example below shows the three windows displayed by MakeInstall as it loads segments onto a floppy diskette. The MakeInstall tool does not generate any command window output. The Seg1 sample Software segment is used in the example. These windows should appear in the order shown below.

```
*****
MakeInstall -p d:\tmp Seg1
```

Make Install

Please enter the following information for the Tape Header:

Name: Developer

Serial: 1

Comment: Test Load

OK Cancel

